Stochastic modeling, analysis and verification of mission-critical software and business processes

By Simos Gerasimou[1], George Mason[1], Colin Paterson[1], Alec Banks[2], Radu Calinescu[1], Daniel Kudenko[1] and Stephen Rowe[2]

[1]Department of Computer Science, University of York, UK
[2]Defence Science and Technology Laboratory, Ministry of Defence, UK

Abstract

Software and business processes used in mission-critical defence applications are often characterised by stochastic behaviour. The causes for this behaviour range from unanticipated environmental changes and built-in random delays to component and communication protocol unreliability. This paper overviews the use of a stochastic modelling and analysis technique called quantitative verification to establish whether mission-critical software and business processes meet their reliability, performance and other quality-of-service requirements.

1. Introduction

Modern military systems are intrinsically complex and sophisticated, typically comprising multiple components with heterogeneous architectures that must operate seamlessly to achieve their goals. Many systems within this domain are mission-critical and must comply with strict dependability, performance and other Quality-of-Service (QoS) requirements. Typical military missions include deployment of unmanned military vehicles for surveillance and battle-space characterisation, and management of underwater, sea and aerial vehicles through a ship's operations room (e.g. a Type 45 operations room). Failing to ensure the system compliance with its QoS requirements could lead to catastrophic consequences, including severe environmental damage, significant financial loss, or loss of human life.

The development of new military systems is typically guided by strict procedures aiming to assure that the operation of these systems conforms to international safety and security standards. For instance, assurance cases are used to provide (Defence Standard 00-56, 2007):

> "*a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment*".

Despite the strict procedures followed during the entire life cycle of software systems, stochastic behaviour is an inherent characteristic of these systems. In autonomous systems, for instance, components may be prone to failure, and communication protocols used for message transmission may be unreliable. Likewise, the processes executed within socio-technical systems are well defined, yet operator behaviour may depart from that prescribed due to oversimplification in the modelling phase and unanticipated environmental factors.

In this paper, we describe how a mathematically based technique called *quantitative verification* can be used to obtain QoS assurance evidence for these types of software and business processes. Quantitative verification (Kwiatkowska, 2007) is a technique for modelling and analysing systems that exhibit stochastic behaviour. In particular, it is used to analyse QoS attributes such as correctness, reliability, response time, energy consumption and resource utilisation. The application of the technique involves constructing finite state-transition models of the analysed system. The states of these models correspond to system configurations that are relevant for the analysed QoS attributes, and the transitions are associated with the possible changes between these configurations. The transitions are annotated with probabilities or transition rates, allowing the specification of discrete-time Markov chains and Markov decision processes, or continuous-time Markov chains, respectively. Quantitative verification uses these models to establish QoS properties of the system specified formally in temporal logic extended with probabilities and costs/rewards. Examples of such properties include the probability that a fault occurs within a specified time period, the expected response time of a service under a given workload, and the expected energy usage of the sensing system of an unmanned vehicle.

Quantitative verification is traditionally used in off-line settings (Norman and Parker, 2014). In these settings, the technique can evaluate the performance-cost trade-offs of alternative system designs or establish if existing systems comply with their QoS requirements. A runtime variant of quantitative verification has been recently proposed by Calinescu *et al.* (2012). This variant supports the continual analysis of autonomous critical systems, to identify and recover from requirement violations or, in some cases, to predict and prevent such violations.

In describing the application of quantitative verification and its runtime variant in the defence domain, we focus on two prevalent types of military systems. First, we cover autonomous systems, which are capable of modifying their behaviour in response to changes in operational context, environment and requirements. Second, we look at socio-technical systems, in which human operators execute decision making processes. We use realistic examples to illustrate how quantitative verification can support the modeling, analysis and verification of key aspects of mission-critical systems and processes. We also explain how the technique can be used to estimate system performance, suggest improvements and find weaknesses of such systems. Finally, we describe how runtime quantitative verification can drive dependable adaptation in autonomous systems.

2. Preliminaries

The use of stochastic state transition models allows for the verification properties specified in formal logics. Markovian models have been described as "the simplest mathematical models for random phenomena evolving in time" (Norris, 1998).

2.1. *Discrete Time Markov Chains (DTMCs)*

A Discrete Time Markov Chain (DTMC) describes a state transition system in which future states of the system are dependent only on the current state and the system is hence considered memoryless. Transitions between states are labelled with probabilities that reflect the stochastic nature of the process. A DTMC can be described formally as a tuple (Baier et al., 2008):

$$\mathcal{M} = (S, \boldsymbol{P}, s_0, AP, L) \tag{2.1}$$

where

- S is a countable, nonempty set of states
- $\boldsymbol{P}: S \times S \rightarrow [0,1]$ is the transition probability function such that for all states s such that $\sum_{s' \in S} \boldsymbol{P}(s,s') = 1$
- s_0 is the initial state and
- AP is a set of atomic propositions and $L : S \rightarrow 2^{AP}$ is a labelling function.

The memoryless nature of the system is defined such that

$$Prob\{X_{n+1} = s_{n+1} | X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0\} = Prob\{X_{n+1} = s_n | X_n = s_n\} \tag{2.2}$$

where $X_0, X_1, ..., X_n$ are successive observations of the system at time steps $0, 1, ..., n$ for all n and possible states s_n. A path π over $\mathcal{M}$ is a possibly infinite sequence of states S such that for any adjacent states s and s' on π, $\boldsymbol{P}(s,s') > 0$.

2.2. *Markov Decision Processes (MDPs)*

Markov Decision Processes extend DTMCs in the sense that they model probabilistic systems that also exhibit nondeterministic behaviour. MDPs model systems in which time progresses in discrete time steps and in any state a nondeterministic choice between probability distributions made.

Formally, an MDP is defined over a set of atomic propositions AP as a tuple

$$\mathcal{M} = (S, s_0, Act, Steps, AP, L) \tag{2.3}$$

where the DTMC is extended such that:

- Act is a set of actions
- $Steps : S \times Act \rightarrow Dist(S)$ is a (partial) probabilistic transition function with $Dist(S)$ denoting the set of all probability distributions over the set S.

In contrast to a DTMC, where the successor state is determined in accordance with the distribution probability over state s, i.e., $P(s,s')$ for $s \in S$, in an MDP a nondeterministic decision is taken based on $|Steps(s)|$ probability distributions. The evolution of an MDP from state s requires that initially an action a is selected nondeterministically from the set of enabled actions $A(s)$ and thereafter, the transition to the next state is chosen randomly, according to the probability distribution $Steps(s,a)$.

2.3. *Continuous Time Markov Chains (CTMCs)*

Time in the discrete Markov chain is abstract and transitions between states may be considered to occur at discrete moments in time. These may be clock ticks of a hardware system or an arbitrary unit of time decided by the modeller. The Continuous Time Markov Chain provides a representation for continuous time models. The CTMC retains the memoryless property of the DTMC (2.2) such that for all states (s_n) and for any sequence $t_0, t_1, ..., t_n, t_{n+1}$ where $t_0 < t_1 < ... < t_n < t_{n+1}$,

$$Prob\{X(t_{n+1}) = s_{n+1} | X(t_n) = s_n, X(t_{n-1}) = s_{n-1}, ..., X(t_0) = s_0\} = Prob\{X(t_{n+1} = s_{n+1} | X(t_n) = s_n\} \tag{2.4}$$

A CTMC over an atomic proposition set AP is defined as a tuple

$$\mathcal{M} = (S, \boldsymbol{Q}, s_0, AP, L) \tag{2.5}$$

where $\boldsymbol{P}$ has been replaced by $\boldsymbol{Q}$, the transition rate matrix.

The element q_{ij} represents the rate at which the system transitions from state i to state j. The diagonal entry q_{ii} is then calculated such that the row sum is zero (Norris, 1998).

Transitions between two states, i and j can happen if $r_{ij} > 0$ and the probability of a transition within t time units is $(1 - e^{r_{ij}t})$. The exponential distribution is used as it is the only continuous distribution that exhibits the memoryless property and fits well the observed behaviour of many systems.

An exponential distribution has a cumulative distribution function as given in (2.6). The parameter λ indicates the number of arrivals expected within a time unit. The inter-arrival time is therefore given by λ^{-1}.

$$f(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

2.4. *Probabilistic Temporal Logics*

Temporal Logics provide a formal language for specifying and reasoning about the behaviour of the model over time. It extends propositional logics with temporal operators and allows system properties to be checked by a model checker.

2.4.1. *Probabilistic Computation Tree Logic (PCTL)*

To verify a DTMC or MDP model the PCTL language (Hansson and Jonsson, 1994) may used to express properties precisely using the syntax

$$\phi ::= true \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \boldsymbol{P}_{\bowtie p}(\varphi) \tag{2.7}$$

where $a \in AP$, φ is a path formula, $p \in [0, 1]$ and $\bowtie \in \{<, >, \leq, \geq\}$. PCTL path formulae are formed as

$$\varphi ::= X\ \phi \mid \phi_1 U \phi_2 \mid \phi_1 U^{\leq n} \phi_2 \tag{2.8}$$

where $n \in \mathbb{N}$. PCTL is enhanced with a satisfaction relation $\vdash$ over the states S and the paths $Paths^{\mathcal{M}}(s)$,$s \in S$. Thus $s \vdash \phi$ means "ϕ is statisfied in state s".

2.4.2. *Continuous Stochastic Logic*

Continuous Stochastic Logics (Aziz, 1996) is a language for specifying the properties of CTMCs and allows for the verification of transient and steady state properties. The state formulae of CSL are defined as

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg\phi \mid S_{\bowtie p}(\phi) \mid P_{\bowtie p}(\varphi) \tag{2.9}$$

and the associated path formulae is

$$\varphi ::= X\phi \mid \phi_1 U \phi_2 \mid \phi_1 U^I \phi_2 \tag{2.10}$$

Of particular note are the time bounded until operator U^I, $I \subseteq \mathbb{R}_{\geq 0}$ and the steady state operator S.

2.5. *Extending Models with Rewards*

Markov models can be annotated with reward/cost structures, in order to increase the range of properties expressible in formal logics. A reward structure for a Markov model $\mathcal{M}$ is a pair of functions (ρ, ι) where:

- $\rho :\rightarrow \mathbb{R}_{\geq 0}$ is the state reward function.
- $\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition reward function

The probabilistic temporal logics are augmented with a reward operator R to enable the analysis and verification of reward-related properties (Andova et al. 2004).

2.6. *Quantitative Verification*

The analysis carried out to establish if one of the stochastic state transition models described in the previous sections satisfies a quantitative property specified in probabilistic temporal logic is called quantitative verification (Kwiatkowska, 2007). Quantitative verification is typically performed automatically by tools called probabilistic model checkers. Widely used probabilistic model checkers include PRISM (Kwiatkowska et al., 2011) and MRMC (Katoen et al., 2011).

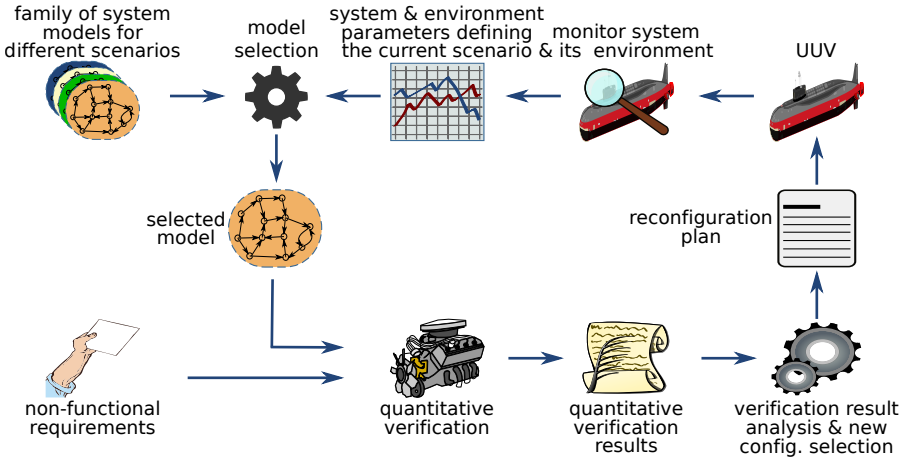


FIGURE 1. Runtime quantitative verification used to implement the control loop of a self-adaptive system

3. Applications

In this section we illustrate the use of quantitative verification using two prevalent types of military systems, i.e., autonomous and socio-technical systems. We also present how combining the technique with reinforcement learning can produce assurance evidence regarding the safety of the selected policy.

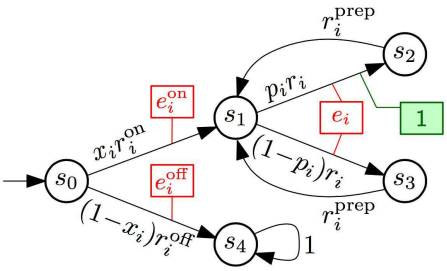
3.1. *Self-Adaptive Systems*

Calinescu *et al.* (2012) advocate the use of quantitative verification at runtime as a way to extend the application of the technique to self-adaptive systems. Originally introduced in (Calinescu and Kwiatkowska, 2009), (Epifani et al., 2009) and further refined in (Calinescu et al., 2011), (Filieri et al., 2011), this approach involves integrating quantitative verification into the closed control loop of self-adaptive systems.

As shown in Fig. 1, the approach requires the continual monitoring of the self-adaptive system and its environment, to identify relevant changes and to quantify them using fast on-line learning techniques. These observations are used to continually update a stochastic model of the system, starting from an initial model provided by the system developers. Probabilistic model checking performed at runtime is then used to re-verify the compliance of these updated models with QoS requirements related to the system response time, reliability, cost, etc. If QoS requirement violations are identified or (when the functionality associated with the unsatisfied requirement has not been exercised) predicted, the results of the analysis support the synthesis of a reconfiguration plan. Executing this plan ensures that the self-adaptive system will continue to satisfy its QoS requirements despite the changes identified during monitoring. The approach has been applied in areas including service-based systems, cloud infrastructure management, embedded and robotic systems and dynamic power management (Epifani et al., 2009), (Calinescu et al., 2011), (Calinescu et al., 2014), (Calinescu and Kwiatkowska, 2009), (Johnson et al., 2013), (Calinescu et al., 2015), (Gerasimou et al., 2014).

We illustrate the use of quantitative verification at runtime using an example based on an unmanned underwater vehicle (UUV) equipped with $n \geq 1$ sensors that can measure the same attribute of the marine environment (e.g., current, salinity or thermocline). Suppose that the i-th sensor takes measurements with a variable rate r_i and consumes energy e_i for each measurement. Every measurement is followed by operations that prepare the sensor for the next measurement, and these operations are carried out with a rate r_i^{prep}. The probability p_i that a measurement is accurate depends on the configurable UUV speed $sp \in [0, 5\text{m/s}]$: $p_i = 1 - \alpha_i sp$, where $\alpha_i \in (0, 0.15)$ is a sensor-specific accuracy factor. Finally, the sensor can be switched on and off in order to save energy ($x_i = 1$ when the sensor is operational and $x_i = 0$ when the sensor is switched off). However, switching the sensor on or off consumes an amount of energy e_i^{on} or e_i^{off}, respectively.

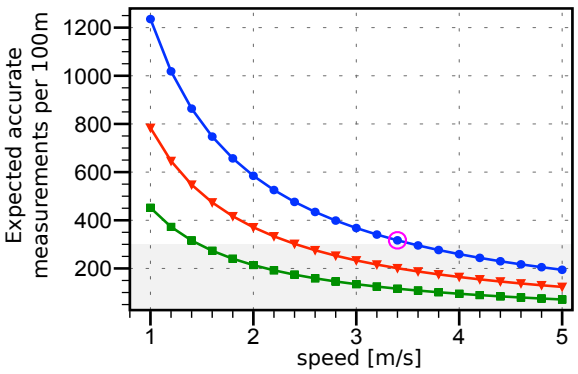
Fig. 2a shows the CTMC model of sensor i-th, adapted from (Gerasimou et al., 2014). From the initial state s_0, the model transitions to state s_1 if the sensor is switched on, or to state s_4 if the sensor is switched off, with rates r_i^{on} and r_i^{off}, respectively. The CTMC models a session during which the sensor is either operational or switched off, so a switched-off sensor remains in state s_4 indefinitely. In contrast, an operational sensor takes measurements with rate r_i and therefore leaves state s_1 with this rate. The next state is either s_3, if the measurement was accurate (with probability p_i), or s_3 otherwise. Irrespective of whether the measurement was successful or not, the model transitions back to state s_1 with the rate r_i^{prep} associated with the operations that prepare the next measurement. The CTMC transitions associated with sensor operations that consume energy are annotated with the energy used by these operations (shown in red/non-shaded squares in Fig. 2a). Similarly, the transition that corresponds to a successful (i.e., accurate) measurement is annotated with a *reward* of 1



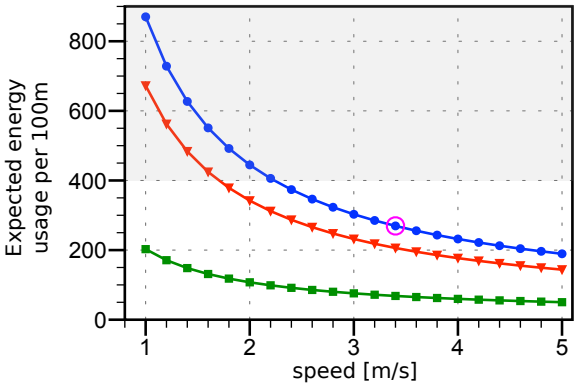
(a) CTMC model of *i*-th UUV sensor

```
ctmc

//sensor operation rates
const double ron_i=10;
const double roff_i=20;
const double rprep_i=100;
const double r_i=5;

//configurable sensor and UUV parameters
const int x_i;
const double sp;

//measurement accuracy
const double alpha_i=0.05;
const double p_i=1-alpha_i*sp;

module sensor
  s : [0..4] init 0;

  [start] s=0 -> x_i*ron_i:(s'=1) +
                 (1-x_i)*roff_i:(s'=4);
  [succ]  s=1 -> p_i*r_i:(s'=2);
  [fail]  s=1 ->(1-p_i)*r_i:(s'=3);
  [prep]  s=2 -> rprep_i:(s'=1);
  [prep]  s=3 -> rprep_i:(s'=1);
  [stop]  s=4 -> (s'=4);
endmodule

rewards "accurate"
  [succ] true : 1;
endrewards

rewards "energy"
  [start] true: (x_i=1)?10:2.4;
  [succ]  true: 0.4;
  [fail]  true: 0.4;
endrewards
```

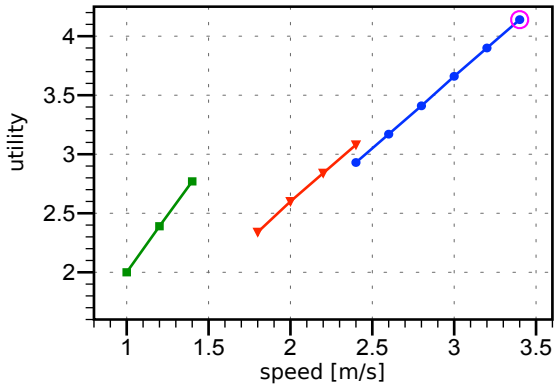
(b) PRISM code for the sensor CTMC model



(c) Verification results for requirement R1



(d) Verification results for requirement R2



(e) Verification results for utility of valid configurations

FIGURE 2. Modelling and quantitative verification of UUV sensor properties

(shown as a green/shaded square in Fig. 2a). The representation of the sensor CTMC model in the high-level modelling language of the probabilistic model checker PRISM (Kwiatkowska et al., 2011) is shown in Fig. 2b. Suppose that the UUV system must comply with the following QoS requirements:

R1 (*Performance*) : "At least 300 accurate measurements must be taken per 100m travelled by the UUV."

R2 (*Energy use*) : "At most 400 Joules must be used by the sensors per 100m travelled by the UUV."

R3 (*Utility*) : "Subject to R1 and R2 being met, the UUV must use a configuration that maximises $utility(x_1, x_2, \ldots, x_n, sp) = w_1 sp + w_2/E$, where E is the energy used by the sensors per 100m travelled by the UUV, and weights $w_1, w_2 > 0$ express the desired trade-off between mission completion time and battery usage.

These requirements are formally specified in CSL as follows:

R1 (*Performance*) : $R^{\text{"accurate"}}_{\geq 300}[C^{\leq 100/sp}]$,

R2 (*Energy use*) : $R^{\text{"energy"}}_{\leq 400}[C^{\leq 100/sp}]$,

R3 (*Utility*) : find argmax $utility(x_1, \ldots, x_n, sp)$ such that **R1** $\wedge$ **R2**

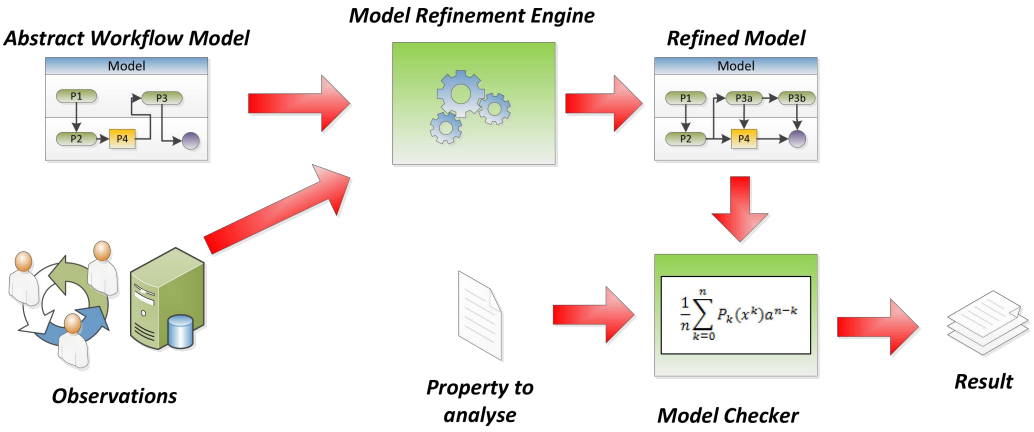


FIGURE 3. Model refinement and continual verification approach

The UUV is required to adapt to changes in the sensor measurement rates $r_1, r_2, \ldots, r_n$ and to sensor failures by dynamically adjusting its speed sp and the set of active sensors such that the three requirements are met.

Fig. 2c,d depict the quantitative verification results obtained for a UUV with $n = 2$ sensors with current measurement rates $r_1 = 5\text{s}^{-1}$ and $r_2 = 9\text{s}^{-1}$. These results represent the expected number of accurate measurements and the expected energy consumption for each 100m travelled, respectively, as a function of the UUV speed. The shaded areas from Fig. 2c,d correspond to parameter values (i.e., sensor configurations) that violate requirements R1 and R2, respectively. These configurations are discarded and the utility of the remaining valid configurations is computed, as depicted in Fig 2e; these results were obtained using $w_1 = 1$ and $w_2 = 200$. The configuration maximising the system utility, circled in Fig. 2c–e, is used to reconfigure the UUV system.

3.2. *Operational Processes*

Traditionally, the modelling of operational processes relies on knowledge of the organisation to describe workflow models that specify the order in which tasks should to be performed. These models are typically oversimplified (van der Aalst, 2010) and hence discrepancies exist between the theoretical model and the actual workflow enacted. To more accurately represent the actual process, workflow mining (van der Aalst, 2003) may be used to extract data from system logs to infer models that may provide insight into the underlying operation of business processes. Where operational processes are instrumented through data logging, Markov models may be inferred using incremental model construction. In this way, states are added to the model as they are encountered and transition probabilities are approximated by the frequencies of the transitions observed (Ghezzi *et al*, 2014).

This modelling allows for formal verification of business processes, with surveys of the state of the art by Morimoto (2008) and subsequently by Groefsema and Bucur (2013). These surveys show how business processes have been verified for a range of functional properties including soundness and consistency. Work by Mendt *et al.* (2011) extended the verification of business process models to include probabilistic model checking, and presented results showing how variability and uncertainty in the process model may used to verify non-functional properties (i.e. quality loss) of the system. Whilst verification of the functional and non functional properties of operational processes has been demonstrated in the literature, work to identify adaptive models at run time for such systems remains a research challenge (Redlich *et al*, 2014).

Our approach is depicted in Figure 3. An abstract workflow model and log data are used as inputs to a model refinement engine. By extracting timing information from the observations it is possible to refine the assumed model and hence produce more accurate verification results.

In the remainder of this section we describe our preliminary experiments aimed at understanding the effects of variation on the verification of operational workflow requirements. To this end, we carried out an experiment in which two web services(WS_1 and WS_2), known to exhibit time-variant execution rates r_1 and r_2, respectively, were invoked at one-minute time intervals over a 24-hour period, and their response times were recorded. The web services considered are provided by the national Rail Enquires server and return arrival and departure information for trains in the UK.

The simple continuous-time Markov chain (CTMC) from Figure 4 models the workflow that uses the two web services. The workflow starts with the invocation of WS_1 (state s_1 in the CTMC) followed by the invocation of WS_2 (state s_2). Service WS_2 returns the expected result and the workflow terminates with probability 0.9; however, with probability 0.1, WS_2 needs to be invoked again due to an erroneous result.

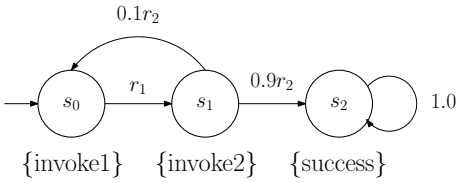
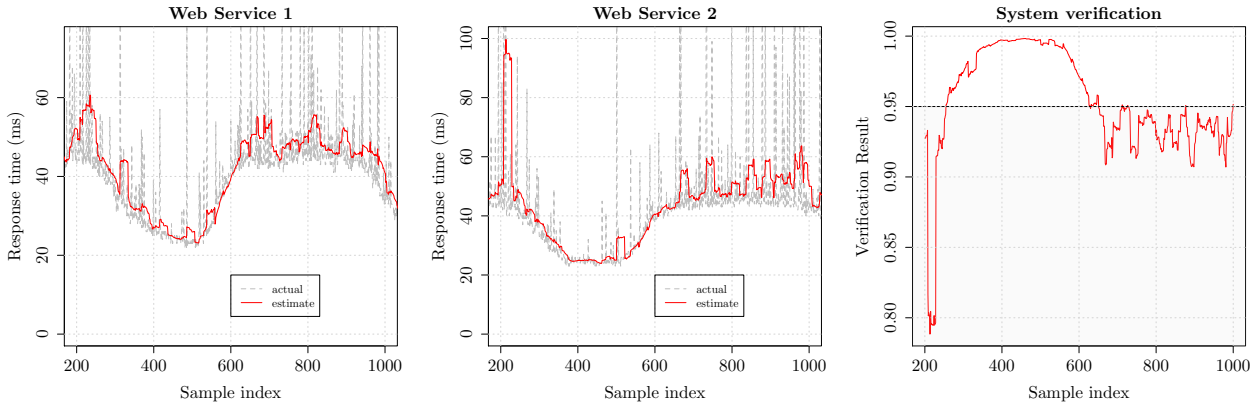


FIGURE 4. CTMC of a simple two-service workflow



a b c

FIGURE 5. Estimation of web service response rates (a, b); and verification results for refined model (c)

A requirement is specified that a request must be completed within 0.25 seconds for 95% of the cases. This requirement may be defined in CSL and verified using the probabilistic model checker PRISM such that

$$P_{=0.95}\ [F^{<=0.25}\ success] \tag{3.1}$$

The mean response time of WS_1 is observed to be 42.79ms and WS_2 is 43.45ms, so the associated rates are $r_1 \simeq 23.37s^{-1}$ and $r_2 \simeq 23.01s^{-1}$. Using the PRISM probabilistic model checker the CTMC system model is verified and the requirement is met such that 96.1% of requests are completed within 0.25 seconds.

Examining the log data for each of the Web Service response times indicates that the response times are subject to periodic variability. In order to account for this variation a simple moving average can be calculated using the most recent n observations. As n increases a smoother approximation is obtained at the cost of responsiveness in the estimator, hence choosing N is non trivial.

Setting $n = 20$ produces response time estimates as shown Fig. 5a,b. At each observation point a PRISM model is then constructed by the refinement engine with estimated transition rates and the property re-verified at that instant. By plotting the probability of completion against time it can be seen that the probability of successfully completing the call within 0.25 seconds varies from a minimum of 78.8% to a maximum of 99.8%. The continual verification results are shown in Figure 5c, where the shaded area corresponds to violations of the requirement (3.1).

It is evident from this that, whilst the single point estimate would lead us to believe that the system is performing adequately, the system fails to meet the requirement specified for a significant amount of time. Static models are therefore unlikely to be adequate for human centric processes in which variability in timing and structure is not itself static.

3.3. *Assured Reinforcement Learning*

Reinforcement learning (RL) is a machine learning technique where a sequential decision making problem is modelled as an (initially unknown) MDP. An autonomous agent navigates through the MDP-modelled environment, and can choose from the set of actions available in its current state. Problem objectives are represented as numerical rewards in the environment and it is the intention of the agent to discover these rewards and therefore a set of behaviours to reach them. The agent discovers rewards through random exploration of the environment states, and remembers the actions it used to get to the reward by associating values based on the reward to the states it traversed to get it, a technique referred to as exploitation. The objective of the agent is to learn a *policy*, i.e. a set of rules for behaviour, that specifies the best action to perform in each state of the environment in order to yield the maximum possible reward from the model. This is known as an *optimal policy* (Wiering and Otterlo, 2012).

A significant body of research has been undertaken to improve the rate at which an optimal policy is found

(Hagen and Kröse, 2003), (Strens, 2000), (Dulac-Arnold et al., 2012), (Stanley and Miikkulainen, 2002). In contrast, there has been minimal work done into ensuring that the learned policy is safe, such that the agent will behave in a manner that is not dangerous to itself or to others. This limitation has prevented RL from being utilized in the safety-critical class of systems where dangerous behaviour can potentially result in catastrophic consequences.

To address this limitation, we are actively working on an approach for *assured reinforcement learning* that uses quantitative verification to ensure that the policy learnt by an RL agent satisfies a given set of probabilistic safety properties. Our approach takes as input this set of properties expressed as PCTL formulae and specifying safety bounds such as *"The probability that the agent enters a failure state must be no greater than 0.02."*. Along with these properties a high-level representation of the problem MDP is required in the form of an *abstract MDP* (AMDP). Note that whereas the original MDP is typically millions or billions of states in size, the AMDP has just a few hundred or thousand (Marthi, 2007). This reduction makes the quantitative verification of AMDP policies feasible.

Accordingly, we plan to use the search-based software engineering techniques from our related work in (Gerasimou et al., 2015) to find AMDP policies that satisfy the required safety properties. The "safe" AMDP policies obtained in this way can then be converted into policy-based reward shaping rules for the RL agent (Sutton and Barto, 1998). This is done by giving the agent a positive reward every time it follows the actions of the selected safe AMDP policy and a negative one otherwise. Our early investigation into how close the achieved RL safety level is to that of the selected safe AMDP policy has yielded encouraging preliminary results.

4. Conclusion

We presented the use of stochastic modelling, analysis and verification to establish key reliability, performance and other quality-of-service properties of mission-critical software and business processes. To this end, we focused on the quantitative verification of Markov chains and Markov decision processes that model the behaviour of these systems that is relevant for the analysed properties. Three application areas were covered in the paper. First, we described the successful application of quantitative verification at runtime within the control loop of self-adaptive software systems. Second, we presenting our ongoing work that uses the technique for the modelling and analyses of operational processes. Finally, we summarised our preliminary work from the area of assured reinforcement learning. Improving the efficiency and capabilities of stochastic modelling, analysis and verification is currently the object of significant effort from the research community. As such, we envisage that many more applications of these techniques will become feasible in the coming years.

Acknowledgments

This paper presents research sponsored by the UK MOD. The information contained in it should not be interpreted as representing the views of the UK MOD, nor should it be assumed that it reflects any current or future UK MOD policy.

REFERENCES

Andova, S., Hermanns, H. & Katoen, J. 2004. Discrete-time rewards model-checked *Formal Modeling and Analysis of Timed Systems*, pages 88–104. Springer.

Aziz, A., Sanwal, K., Singhal, V., & Brayton, R. 1996. Verifying continuous time Markov chains *Computer Aided Verification*, pages 269–276. Springer.

Baier, C. & Katoen, J.P. 2008. *Principles of model checking*. MIT press.

Calinescu R. & Kwiatkowska M. 2009. Using quantitative analysis to implement autonomic it systems. *International Conference on Software Engineering*, pages 100–110.

Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., & Tamburrelli, G. 2011. Dynamic QoS management and optimization in service-based systems. *Transactions on Software Engineering*, 37(3):387–409.

Calinescu, R., Ghezzi, C., Kwiatkowska, M., & Mirandola, R. 2012. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77.

Calinescu, R., Rafiq, Y., Johnson, K., & Bakir, M.E. 2014. Adaptive model learning for continual verification of non-functional properties. *International Conference on Performance Engineering*, pages 87–98.

Calinescu, R., Gerasimou, S., & Banks., A. 2015. Self-adaptive software with decentralised control loops. *International Conference on Fundamental Approaches to Software Engineering*, pages 235–251.

Gerasimou, G., Tamburrelli, G., & Calinescu, R. 2015. Search-Based Synthesis of Probabilistic Models for Quality-of-Service Software Engineering. *30th IEEE/ACM International Conference on Automated Software Engineering*, to appear.

Dulac-Arnold, G., Denoyer, L., Preux, P., & Gallinari, P. 2012. Fast reinforcement learning with large action sets using

error-correcting output codes for MDP factorization. *European conference on Machine Learning and Knowledge Discovery in Databases*, pages 180–194. Springer.

Epifani, I., Ghezzi, C., Mirandola, R., & Tamburrelli, G. 2009. Model evolution by run-time parameter adaptation. *International Conference on Software Engineering*, pages 111–121.

Filieri, A., Ghezzi, C., & Tamburrelli, G. 2011. Run-time efficient probabilistic model checking. *International Conference on Software Engineering*, pages 341–350.

Gerasimou, S., Calinescu, R., & Banks, A. 2014. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 115–124.

Ghezzi, C., Pezzè, M., Sama, M., & Tamburrelli, G. 2014. Mining behaviour models from user-intensive web applications. *International Conference on Software Engineering*, pages 277–287.

Groefsema, H. & Bucur, D. 2013. A survey of formal business process verification: from soundness to variability. *International Symposium on Business Modeling and Software Design*.

Hagen, S. & Kröse. B. 2003. Neural Q-Learning. *Neural Computing & Applications*, 12(2):81–88.

Hansson, H. & Jonsson, B. 1994. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535.

Johnson, K., Calinescu, R. & Kikuchi, S. 2013. An incremental verification framework for component-based software systems. *International Symposium on Component-based Software Engineering*, pages 33–42.

Katoen, J.P., Zapreev, I. S., Hahn, E. M., Hermanns, H. & Jansen, D. N. 2011. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90 – 104.

Kwiatkowska, M. 2007. Quantitative verification: models, techniques and tools. *Joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 449–458.

Kwiatkowska, M., Norman, G. & Parker, D. 2007. Stochastic model checking. *International Conference on Formal Methods for Performance Evaluation*, pages 220–270.

Kwiatkowska, M., Norman, G. & Parker, D. 2011. Prism 4.0: verification of probabilistic real-time systems. *International Conference on Computer Aided Verification*, pages 585–591.

Marthi, B. 2007. Automatic shaping and decomposition of reward functions. *International Conference on Machine Learning*, pages 601–608..

Mendt, T., Sinz, C. & Tveretina, O. 2011. Probabilistic model checking of constraints in a supply chain business process. *Business Information Systems*, pages 1–12. Springer.

Morimoto, S. 2008. A survey of formal verification for business process modelling. *Computational Science*, pages 514–522. Springer.

Norman, G. & Parker, D. 2014. Quantitative Verification: Formal Guarantees for Timeliness, Reliability and Performance. Knowledge Transfer Report, London Mathematical Society and the Smith Institute for Industrial Mathematics and System Engineering.

Norris, J. R. 1998. *Markov chains*. Cambridge University Press.

Redlich, D., Blair, G., Rashid, A., Molka, T., & Gilani, W. 2014. Research challenges for business process models at run-time. *Models@run.time*, pages 208–236. Springer.

Stanley, K. O. & Miikkulainen, R. 2002. Efficient reinforcement learning through evolving neural network topologies. *Genetic and Evolutionary Computation Conference*, pages 569–577.

Strens, M. 2000. A Bayesian framework for reinforcement learning. *International Conference on Machine Learning*, pages 943–950.

Sutton, R. S., & Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press.

UK Ministry of Defence. 2007. Defence Standard 00-56, Issue 4: Safety Management Requirements for Defence Systems.

van der Aalst, W. M., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G. and Weijters, A. J. 2003. Workflow mining: A survey of issues and approaches. *Data & knowledge engineering*, 47(2):237–267.

van der Aalst, W. M. 2010. Business process simulation revisited. *Enterprise and Organizational Modeling and Simulation*, pages 1–14.

Wiering, M., & Otterlo, M. 2012. Reinforcement learning and Markov decision processes. *Reinforcement Learning: State-of-the-art*, volume 12, pages 3–42. Springer.